International Journal of Advances in Engineering and Management (IJAEM)
Volume 6, Issue 11 Nov. 2024, pp: 118-127 www.ijaem.net ISSN: 2395-5252

Multi-Tenant yet Customizable Cloud Native SaaS Web Application leveraging AIML: Architecture & Strategies

Sunil S. Bilur[1], Mahesh S. Salunkhe[2]
[1]*Student, Kolhapur Institute of Technology's, College of Engineering, Kolhapur, India.*
[2]*Associate Professor, Kolhapur Institute of Technology's, College of Engineering, Kolhapur, India*.

Date of Submission: 01-11-2024 Date of Acceptance: 10-11-2024

ABSTRACT:The rise of cloud-native architectures has enabled the rapid development and deployment of scalable Software-as-a-Service applications. However, building multi-tenant systems that support extensive customization for individual tenants remains a challenge. This paper explores strategies to overcome these challenges in the context of developing an AI/ML-enhanced Enterprise Resource Planning (ERP) system for educational institutions. We propose a microservice-based architecture that decouples AI/ML models from the main application, dynamically generating UI components, forms, routes, and implementing flexible role-based access control (RBAC). This architecture allows tenant-specific customization without sacrificing the scalability, security, and maintainability of a cloud-native system. Real-world implementation details and strategies for sustainability are discussed, along with challenges faced.

KEYWORDS: Multi-tenant, Customizable, Cloud Native, Sofware-as-a-service (SaaS), Web Application, Software Architecture, Artificial Intelligence, Machine Learning

I. INTRODUCTION

Cloud-native systems have revolutionized how SaaS applications are developed, offering scalability, flexibility, and lower operational costs. However, the shift towards multi-tenancy—where multiple clients i.e. tenants share a single and common instance of the software—often leads to challenges when attempting to provide customizations specific to each tenant. Multi-tenant systems tend to prioritize uniformity across tenants to maintain simplicity, but in domains like education, the need for tailored solutions is paramount. Educational institutions often require customizable workflows, user interfaces (UIs), and role-based access control (RBAC) to fit their specific operational requirements.

The architecture of a multi-tenant application which provides significant tenant-specific customization while ensuring system scalability and maintainability can be complex. Furthermore, deploying Artificial Intelligence and Machine Learning (AIML) models as part of the SaaS application adds another layer of complexity. AIML services often require specific computational resources and cannot easily coexist within the same service as the main application. This paper presents strategies to address these challenges through dynamic component generation, microservices for AIML, and flexible role-based access control.

Research Contributions

This paper contributes the following things to the body of knowledge:

a) Proposing a microservices inspired architecture that decouples AIML models from the main application service which allows for its independent scaling.
b) Introducing methods to dynamically generate elements of web application like forms, routes, and UI components to enable tenant-specific customizations while maintaining scalability.
c) Developing a flexible, customizable RBAC system where roles and permissions are dynamically generated allowing to define a new role and assign features to it.
d) Offering sustainability strategies to ensure that customization does not impact scalability and performance in cloud-native environments.

II. LITERATURE REVIEW

Multi-tenant SaaS application development has been the subject of extensive research. This section reviews existing work in these areas,

International Journal of Advances in Engineering and Management (IJAEM)
Volume 6, Issue 11 Nov. 2024, pp: 118-127 www.ijaem.net ISSN: 2395-5252

particularly around customizability in multi-tenant systems. Although efficient in resource utilization, multi-tenant systems are inherently less flexible when it comes to tenant-specific customizations. Research on multi-tenancy and SaaS primarily focuses on scalability, resource allocation, and tenant isolation, but not much attention has been given to customizability at the tenant level.

M. A. Rothenberger and M. Srite have investigated why some enterprise ERP systems have high level of customization despite generally accepted best-practice of limiting customization [1]. Also, Hansen FH, Haddara M, Langseth M. have surveyed the research on ERP system customization [2]. F. Aslam has discussed the benefits and challenges of customization within SaaS cloud solutions [3].

Qasem Ali A, Abd Ghani AA, Md Sultan AB, Zulzalil H. conoducted an investigation in an empirtical manner to study impact of software customization on SaaS [4]. Ali, Hazura, Abdulrazzaq Qasem and Md Sultan, Abdul Azim and Zulzalil, Abu Bakar and Abd Ghani, have delved into challenges related to customization in SaaS, and then have mapped these challenges in order to structure the study of SaaS customization [5]. W. Sun, X. Zhang, P. Sun, C. J. Guo and H. Su have explored issues and challenges faced by SaaS vendors while doing customization and configuration [6].

R. Mietzner, F. Leymann, K. Pohl and A. Metzger have discussed variability modeling techniques from the field of software product line engineering and how it can support SaaS providers to manage variability and requirements in SaaS application, they have also discussed orthogonal variable model (OVM) [7]. N, Kurono and M. Aoyama have extended the OVM model for metadata driven architecture for multi-tenat cloud apps or services [8].

No matter what method is used to represent the variability along with commonality, the customization of saas for each tenant still remains a challenge. A. B. M. Sultan, A. Q. Ali, H. Zulzalil and A. A. A. Ghani have adopted a mapping approach which systematic to study and investigatesolutions that solve the SaaS customization problems [9]. Various frameworks have been proposed which supposedly ease the development of SaaS. M. Choi and W. Lee have proposed a web application framework to develop SaaS applications [10]. Microservices architectures have been widely adopted to improve scalability, modularity, and maintainability of cloud-native applications. Techniques related to microservices have been researched for customization of SaaS in multi-tenant environment. H. Song, F. Chauvel and P. Nguyen have studied and simplified the key concepts in customization of multi-tenant systems. They have also described high-level principles and design with a reference architecture [11].

L. P. Tizzei, R. F. G. Cerqueira, M. Nery and V. C. V. B. Segura have illustrated development of multi-tenant SaaS applications using microservices and software product line techniques [12]. P. H. Nguyen, H. Song, F. Chauvel, J. Glattetre and T. Schjerpen have presented experimental findings of the design of a novel cloud-native architecture where they have customized multi-tenant SaaS using microservices [13]. H. Song, A. Solberg and F. Chauvel have propsed a novel and fresh architectural style to do customization of SaaS using microservices which are intrusive in a deep manner [14].

M. Makki, S. Walraven, D. Van Landuyt and W. Joosen have presented a workflow customization that makes it possible for application vendors to keep mult-tenant customization concerns and workflow design loosely coupled which results into better manageability, the tenant preferences are activated at runtime without sacrificing the scalability and manageability [15]. M. Makki, B. Lagaisse, W. Joosen and D. Van Landuyt have analyzed and compared how the different customization of business process strategies has implications on the multi-tenant SaaS application [16].

III. HIGH LEVEL ARCHITECTURE

The system accounting for scalability and customizability in a multi-tenant application which is discussed in this paper is shown is shown in fig. 1. It is divided into two main parts as follows,

a) Main application
b) AIML service

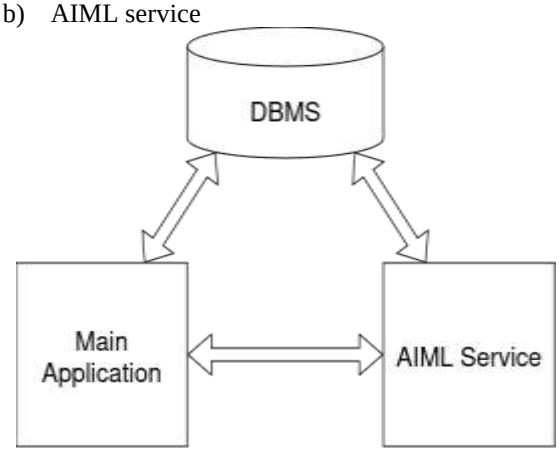


Fig. 1

International Journal of Advances in Engineering and Management (IJAEM)
Volume 6, Issue 11 Nov. 2024, pp: 118-127 www.ijaem.net ISSN: 2395-5252

We are not concerned with the internal architecture of both main application and AIML service. Internally, they can be monolithic or microservices but at top level these two parts are like two services in a microservice architecture which are deployed independently.
They communicate to each other over API. Only signals are communicated over the API, Exchange of large of amount of data over API is discouraged. The database used by main application is shared by AIML service.

Multi-Tenant Architecture Overview

A multi-tenant system serves multiple clients by running a single instance of the application. Tenants share the same codebase and database schema, although each tenant's data is logically isolated. This architecture allows for efficient resource utilization but typically limits the level of customizability for individual tenants. In this ERP system, tenants require highly customizable interfaces, leading to the need for more flexible solutions.

Customization Challenges in Multi-Tenant Systems

The core challenge of multi-tenant systems is balancing scalability with customization. Adding excessive customizability can strain system performance, create security risks, and introduce maintainability issues. It is not easy to offer multi-tenant SaaS applications which are highly customizable. This is the reason SaaS offerings are generally designed to fit all users with a single approach [17]. The key idea we have discussed in this paper is to dynamically generate tenant-specific components (e.g., user interface, forms, routes, etc.) and use a role-based access control (RBAC) system where roles are arbitrarily defined and permissions are checked at runtime based on tenant needs.

IV. CUSTOMIZATION APPROACH

The essence of being able to do customization in multi-tenant environment without sacrificing manageability and without changing the core of the application is dynamic generation of as many elements of the application as possible. This allows scalable and manageable development of multi-tenant customizable SaaS application. The dynamic generation is done based on some declarative configuration or metadata.

All of the declarative configuration and metadata is stored in tenant's database. Based on the architecture database switching might need to be handled. If the app is server side rendered the dynamic generation of required HTML is done on server side, else if the app is client side rendered then the frontend receives the declarative configuration and based on it renders the frontend HTML. Here frontend acts as an interpreter that interprets the dynamic declaration and generates the result.

Note that depending on the implementation technology some things are technically harder or cumbersome to implement in one that in the other. For example, dynamic naming of routes is much easier to implement in client side rendered technologies like MEAN stack compared to something which is server side rendered form example in PHP

Dynamic User Interface and Navigation

The user interface can be defined largely as a combination of navigation structure and page layout shown for a particular link from the navigation structure. The frontend is developed as a set of components which get shown in the page. The page layout is a grid which arranges the components dynamically in the grid inside a structure of rows and columns which are responsive.

This allows tenants to have different navigation structure and each link from navigation structure can have any set of components arranged dynamically based on dynamic configuration which defines components and how they are arranged into rows and columns in the grid. The routes for each link in the navigation structure are also generated dynamically based on the declarative configuration.

Dynamic Form Generation

Form are one of the main elements of SaaS web application. It is highly likely that each tenant might need to input different data based on different requirements and would like to customize forms. In this case, it helps to generate form structure using a declaration which lists out the form layout and other information such as field types. Based on the declaration stored in tenant's database the form is dynamically constructed at runtime. This method ensures that each tenant can define unique forms tailored to their need without required changes to the core database.

Dynamic generation of the form to display to the tenant on the frontend is only part of the story. The data sent form the frontend via forms needs to be stored in database. Since, the forms are dynamically generated and can have any number of fields having a schemeless document database such as MongoDB, CouchDB etc. is of added advantage here and eases the handling of schemeless nature of data generated from the forms which needs to be stored at the backend.

International Journal of Advances in Engineering and Management (IJAEM)
Volume 6, Issue 11 Nov. 2024, pp: 118-127 www.ijaem.net ISSN: 2395-5252

Dynamic Roles and Access Control

As we discussed user interface, routes, navigation and components grid are dynamically generated and their declarative configuration is grouped under particular role. A role is basically a collection of these declarative configuration. It is like a glue that binds together all the other important configurations that allow for declarative customization for that particular role. If we want to add a new role, we create a new group of declaration that declares and groups together user interface, routes, navigation and components grid under a role.

Access control has been implemented per component basis. This gives fine grained control on access control. A declarative configuration is declared for each component that gets loaded in page layout. This declarative configuration tells what part of the component the defined role has access to. The configuration also lists out any variation that needs to implemented for a particular role for the given component.

The declarative configuration for access control is tightly coupled with the user interface. The idea behind making the access control tightly coupled with user interface is that, a lot of times if a user does not have access to something it will not be shown in user interface or will be shown but in disabled mode so access control is naturally tightly coupled with user interface.

Binding pieces together and runtime generation approach to customization

The effect of this approach towards customization is the flexibility we get in defining the customization for each tenant. If an application is developed in this manner, we can define any user interface having different page layout of different features for any role which are again arbitrarily defined with any access control and variation at component level.

To summarize, Customization is provided through,

a) Binding various declarative configurations together under a role defining the user interface and thus features that the role will get access to.
b) Runtime generation of elements of the system.

V. MICROSERVICES FOR AIML INTEGRATION

Decoupling AIML Deployment from main application

Deploying AIML models within the same monolithic service as the core application can lead to resource contention and scaling challenges. To resolve this, we implemented a microservice architecture where AIML models are deployed in isolated services. These services are responsible for processing tenant-specific requests, such as sentiment analysis on feedback forms or predictive analytics for student performance. Microservice architecture is used mainly to decouple the AIML models from the core application. By doing this, we reduce the complexity of the system and ensure that AIML features can be updated or scaled independently. Also, Microservices can simplify development and project management, and can eliminate the need for separate operations teams.

Programming Language Agnostic and Leveraging Python community for AIML

In a microservices-based application, developers can connect services written in different programming languages and deployed on various platforms. This flexibility allows teams to choose the languages and technologies that best suit their project requirements and expertise.

By being language-agnostic, teams can swiftly embrace new technologies as they evolve, no longer tied to a single technology stack, and free to use the optimal tools for each task. E.g. you can use high performant Rust backend for main application and leverage huge python community for data science and AIML to develop AIML service in Python language.

Triggering AIML Services & Event-Driven Model

The main application triggers AIML services through API calls. The AIML microservices are triggered by events from the main application service. For example, when a new feedback form is submitted, an event is fired to the AIML service to analyse sentiment. This ensures that AIML processing does not block the core application's performance and can be handled asynchronously. The decoupling of AIML services ensures that the main application remains lightweight and performant, even as AIML models evolve and become more computationally intensive.

Communication between main application and AIML service

The main application and AIML service do not exchange large amount of data directly over network. The API between them is only to contact signals and trigger the events in AIML services. When AIML service gets triggered calculates the result and stores it in the database and this result is fetched by main application from the database. The database is the common ground for these two entities to exchange large amount of data.

International Journal of Advances in Engineering and Management (IJAEM)
Volume 6, Issue 11 Nov. 2024, pp: 118-127 www.ijaem.net ISSN: 2395-5252

VI. CASE STUDY: CUSTOMIZABLE ERP SYSTEM FOR EDUCATIONAL INSTITUTIONS

This paper has been presented in context of developing an ERP system for educational institutions like schools and colleges and thus the techniques presented here are well tested in real world use case.

The ERP application for educational institutions is built using MEAN stack and the decoupled AIML microservice is built in Python using Flask framework. The database used is MongoDB. In traditional sense, the frontend is significantly more complex than the backend. This is because the declarative configuration is sent from the backend to the frontend and frontend interprets this configuration and generates results.

In the application, every user must have an associate role and each role has a role_config declared for it which holds the declarative configuration for that role. A role_config is basically declarative configuration that defines everything about that role, it defines following things

a) Navigation and entries that will be shown to that role
b) Routes for entries in navigation
c) Customizable page layout which is basically a grid of components
d) Variation and access control for any component

A role_config can be declared as per the requirement and assigned to a role. A role can then be associated with users for which we want to have those customization and features defined in role_config. This approach to handling role-based access control makes the system flexible as any component grid with required customization can be rendered for any role/user. Roles are created as per requirement and not defined in advance.

Following is an example of role_config for admin role

```
{
 "role": "admin",
 "pri_nav": [
  {
   "name": "Learning",
   "route": "lms"
  },
  {
   "name": "Notice Board",
   "route": "notices"
  },
  {
   "name": "Exam & Results",
   "route": "exams"
  },
  {
   "name": "Office Activities",
   "route": "office"
  }
 ],
 "sec_nav": {
  "lms": [
   {
    "type": "entry",
    "name": "Dashboard",
    "route": "dashboard",
    "comp": "comp2",
    "icon": "dashboard"
   },
   {
    "type": "entry",
    "name": "Admission",
    "route": "admission",
    "comp": [
     [
      "student-admission",
      {}
     ]
    ],
    "icon": "assign-user"
   },
   {
    "type": "entry",
    "name": "Student Information",
    "route": "student-info",
    "comp": [
     [
      "row",
      [
       "col",
       12,
       [
        "student-info-title",
        {}
       ],
       {}
      ]
     ],
     [
      "row",
      [
       "col",
       12,
       [
        "student-info-control-bar",
        {}
       ]
```

```
        ]
       ],
       [
        "row",
        [
         "col",
         12,
         [
          "student-info-table",
          {}
         ]
        ]
       ]
      ],
      "icon": "assign-user"
     },
     {
      "type": "menu",
      "entries": [
       {
        "name": "Feedback Management",
        "route": "feedback-management",
        "comp": [
         [
          "row",
          [
           "col",
           12,
           [
            "feedback-mgmt-title",
            {}
           ]
          ]
         ],
         [
          "row",
          [
           "col",
           6,
           [
            "feedback-mgmt-schedule",
            {
             "revoke": [
              "new_feedback"
             ]
            }
           ]
          ],
          [
           "col",
           6,
           [
            "feedback-mgmt-analysis",
            {}
           ]
          ]
         ]
        ]
       },
       {
        "name": "Feedback Questions",
        "route": "feedback-questions",
        "comp": [
         [
          "row",
          [
           "col",
           12,
           [
            "feedback-ques-title",
            {}
           ]
          ]
         ],
         [
          "row",
          [
           "col",
           12,
           [
            "feedback-ques-overview",
            {}
           ]
          ]
         ],
         [
          "row",
          [
           "col",
           8,
           [
            "feedback-ques-list",
            {}
           ]
          ],
          [
           "col",
           4,
           [
            "feedback-ques-tags-chart",
            {}
           ]
          ]
         ]
        ],
        "options": "null"
       }
      ],
      "name": "Feedback",
      "id": "feedback-menu",
      "icon": "form"
```

International Journal of Advances in Engineering and Management (IJAEM)
Volume 6, Issue 11 Nov. 2024, pp: 118-127 www.ijaem.net ISSN: 2395-5252

```
    }
   ]
  }
}
```

Explanation for role_config:
At top level in JSON we have

a) role – defines the name of the role
b) pri_nav – First level navigation
c) sec_nav – second level navigation

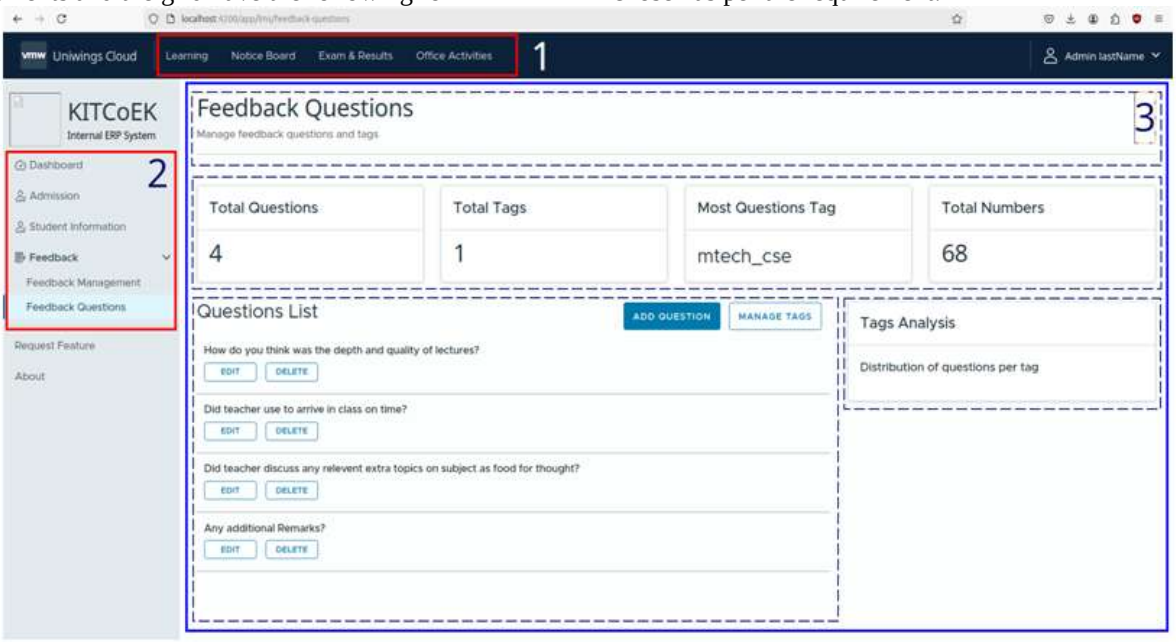
Think of pri_nav as tabs, where each pri_nav will hold the `sec_nav`. The route field in`pri_nav` document is the key name in `sec_nav` i.e. routeparameter is used to link `sec_nav` document to the `pri_nav` document. Essentially it forms a tree like structure of navigation. The entries in `sec_nav` can be either an entry or menu. If it is a menu, it will further contain other entries, else if it is an entry, it is the end of navigation tree. Upon clicking the entry in `sec_nav` a grid of components will be rendered. The grid of component is generated from a declarative configuration which is declared under `comp` key

The format of `comp` key's value:

The components are arranged in a grid. The grid has 12 columns. The grammar of nested elements and the grid have the following form

```
GRID = [ ELEMS ]
ELEMS = ELEM | ELEM ELEMS
ELEM = ["row", ELEMS ]
| ["col", width (max 12), ELEMS ]
|["component_name", "component configuration"]
```

The elements generated from the declarative configuration are highlighted in figure shown at the bottom of this page. Area highlighted by red border and numbered as 1 shows primary navigation which works as tabs and is generated from declarative configuration `pri_nav`. Area numbered 2 is secondary navigation generated form `sec_nav`. There are three entries according to the declaration followed by a feedback menu which contains two entries. Note that changing the order of the entries changes the order in the user interface. Each entry is associated with a grid layout. The generated grid is shown in blue border numbered as 3. The arrangement of components and their associated configuration is done declaratively. This grid is declared under `comp` key in entry's document. The dark blue coloured dashed borders represent the components. Note that the customization parameters for each component like variation and access control are hard coded into thefrontend and any structure of document and parameters can be chosen as per the requirement.



Collections that are important for customizability and management

Institutions – stores the value of all the institutions
It holds tenant id called iid (institution id) which uniquely recognizes the tenant among all the tenants. It also stores other details of tenants like name, short name, year of establishment and so on.

role_configs – stores all roles and their configuration

International Journal of Advances in Engineering and Management (IJAEM)
Volume 6, Issue 11 Nov. 2024, pp: 118-127 www.ijaem.net ISSN: 2395-5252

Perhaps the most important collection, It stores the declarative configuration of each role which decides what gets rendered on the screen, customization and access control of the rendered components

users_current – the current users enrolled in the institution
all the users from users to principal which can log into the system are stored in users_current.

users_old - users which are not currently enrolled in the institution
Even if a user is not associated with the institution, no user is actually permanently deleted

forms – declarative configurations for forms
These forms are dynamically generated at runtime

Collections that represent academic structure of institution

a) academic_programs – stores degree programs e.g. bacherlor of technology, computer science
b) academic_departments – stores departments of institutions, If the institution has no notion of departments like for certain schools in such cases at least one default department document is present in this collection
c) academic_classes – classes groups together a set of students
d) academic_courses – it holds a list of all courses documents which are taught in the institution

These four collections are loosely coupled, in the sense, that they can represent any educational institution whether it is a school or college. The design is chosen like this so that it can represent any organization that has students in it, where courses are taught.
The modules implemented in this ERP are

a) Admission Module
b) Student Information Module
c) Student Feedback Module
d) Leave Management Module
e) Library Management Module

Discussing every module in detail would go out of scope and is not feasible in this paper but the essence of how the customization technique is used in the ERP have been discussed.

All of the modules demonstrate the techniques used for customization so far. Every tenant (educational institution) might choose to input different information at the time of admission from students so it requires different forms for each tenant. Therefore, dynamic generation of forms is implemented here. The declarative configuration that is given to the frontend that decides what fields will be generated is stored in each tenant's metadata collection with kind field given as dynamic form.

Student information module gives information about students. Access control fields implemented revoke_accesss in which if we specify a field then that fields info is not show to that role and all the users having that role

Sentiment analysis has been implemented on student feedback module. When feedback is over a signal is sent to the AIML service, which then processes the input and stores the results into database.

Users can apply for leaves, which go to higher level of authority for approval. Once the leave is approved then the leave is granted to the user and shown to user. History of everything like when was leave applied, when was it approved and by whom was it approved is stored in database and shown to users if they have proper access control.

VII. RESULTS AND DISCUSSION

ERP system, developed for educational institutions, provides a real-world example of these strategies in action. The system supports multiple educational institutions with different user roles, user interface, access control and data models. Each tenant can configure their own user interface for student admissions, course management, and feedback collection. Building a customizable, multi-tenant cloud-native system presents several challenges, particularly around scalability and maintainability. However, by decoupling AIML services and dynamically generating customizable components, we achieved a balance between flexibility and performance. Future improvements include expanding customization to include tenant-specific data models and deeper AIML integration.

VIII. CONCLUSION

Multi-tenant SaaS systems inherently face limitations in customization due to their shared infrastructure, which prioritizes standardization and uniformity across all tenants. This paper explored solutions to the challenges of building a multi-tenant system that supports extensive customization, using the example of an educational ERP system enhanced with AIML models. By leveraging cloud-native technologies, microservices, and dynamic generation of key application components, we demonstrated how it is possible to deliver a highly customizable experience for individual tenants while maintaining scalability and operational efficiency.

One of the key contributions of this research was the development of a microservice architecture that decouples AI and ML models from the main application. This separation not only allows for independent scaling of computationally intensive AIML services but also ensures that updates to models can be made without impacting the core functionality of the application. Additionally, the ability to dynamically generate UI components, forms, routes, and access controls at runtime allows for extensive customization that meets the specific needs of different educational institutions. These customizations do not require changes to the core system, making maintenance simpler and reducing the risk of regressions.

While the presented strategies show significant promise, the development of customizable multi-tenant SaaS systems remains a complex and evolving field. Future work is focused on expanding the scope of customization, potentially enabling clients/tenants to customize the workflows. Additionally, continuous improvements in flexibility of the system and the refinement of access control strategies will be necessary to keep pace with evolving requirements and technological advancements.

REFERENCES

[1]. M. A. Rothenberger and M. Srite, "An Investigation of Customization in ERP System Implementations," in IEEE Transactions on Engineering Management, vol. 56, no. 4, pp. 663-676, Nov. 2009, doi: 10.1109/TEM.2009.2028319.

[2]. Hansen FH, Haddara M, Langseth M. Investigating ERP System Customization: A Focus on Cloud-ERP. Procedia Comput. Sci., vol. 219, Elsevier B.V.; 2023, p. 915–23. https://doi.org/10.1016/j.procs.2023.01.367.

[3]. F. Aslam, "The Benefits and Challenges of Customization within SaaS Cloud Solutions," Amer. J. Data, Inf. Knowl. Manag., vol. 4, no. 1, pp. 14–22, 2023, doi: 10.47672/ajdikm.1543.

[4]. Qasem Ali A, Md Sultan AB, Abd Ghani AA, Zulzalil H. An Empirical Investigation of Software Customization and Its Impact on the Quality of Software as a Service: Perspectives from Software Professionals. Applied Sciences. 2021; 11(4):1677. https://doi.org/10.3390/app11041677

[5]. Ali, A. B. Md Sultan, A. Abd Ghani, and H. Zulzalil, "Customization of Software as a Service Application: Problems and Objectives," J. Comput. Sci. Comput. Math., vol. 8, no. 3, pp. 27–32, Sep. 2018, doi: 10.20967/jcscm.2018.03.001.

[6]. W. Sun, X. Zhang, C. J. Guo, P. Sun and H. Su, "Software as a Service: Configuration and Customization Perspectives," 2008 IEEE Congress on Services Part II (services-2 2008), Beijing, China, 2008, pp. 18-25, doi: 10.1109/SERVICES-2.2008.29.

[7]. R. Mietzner, A. Metzger, F. Leymann and K. Pohl, "Variability modelling to support customization and deployment of multi-tenant-aware Software as a Service applications," 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems, Vancouver, BC, Canada, 2009, pp. 18-25, doi: 10.1109/PESOS.2009.5068815.

[8]. M. Aoyama and N. Kurono, "An Extended Orthogonal Variability Model for Metadata-Driven Multitenant Cloud Services," 2013 20th Asia-Pacific Software Engineering Conference (APSEC), Bangkok, Thailand, 2013, pp. 339-346, doi: 10.1109/APSEC.2013.53.

[9]. Q. Ali, A. B. M. Sultan, A. A. A. Ghani and H. Zulzalil, "A Systematic Mapping Study on the Customization Solutions of Software as a Service Applications," in IEEE Access, vol. 7, pp. 88196-88217, 2019, doi: 10.1109/ACCESS.2019.2925499.

[10]. W. Lee and M. Choi, "A Multi-tenant Web Application Framework for SaaS," 2012 IEEE Fifth International Conference on Cloud Computing, Honolulu, HI, USA, 2012, pp. 970-971, doi: 10.1109/CLOUD.2012.27.

[11]. H. Song, P. Nguyen, and F. Chauvel, "Using Microservices to Customize Multi-Tenant SaaS: From Intrusive to Non-Intrusive," 2019. doi: 10.4230/OASIcs.Microservices.2017/2019.1.

[12]. L. P. Tizzei, M. Nery, V. C. V. B. Segura, and R. F. G. Cerqueira, "Using Microservices and Software Product Line Engineering to Support Reuse of Evolving Multi-tenant SaaS," in Proc. 3rd Int. Workshop Softw. Eng. Smart Cyber-Physical Syst. (SEsCPS), New York, NY, USA, 2017, pp. 22–28, doi: 10.1145/3106195.3106224.

[13]. H. Song, P. H. Nguyen, F. Chauvel, J. Glattetre and T. Schjerpen, "Customizing Multi-Tenant SaaS by Microservices: A Reference Architecture," 2019 IEEE International Conference on Web Services (ICWS), Milan, Italy, 2019, pp. 446-448, doi: 10.1109/ICWS.2019.00081.

[14]. H. Song, F. Chauvel, and A. Solberg, "Deep customization of multi-tenant SaaS using

intrusive microservices," in Proc. 40th Int. Conf. Softw. Eng.: New Ideas and Emerging Results (ICSE-NIER), New York, NY, USA, 2018, pp. 97–100, doi: 10.1145/3183399.3183407.

[15]. M. Makki, D. Van Landuyt, S. Walraven, and W. Joosen, "Scalable and manageable customization of workflows in multi-tenant SaaS offerings," in Proc. 31st Annu. ACM Symp. Appl. Comput. (SAC), New York, NY, USA, 2016, pp. 432–439, doi: 10.1145/2851613.2851627.

[16]. M. Makki, D. Van Landuyt, B. Lagaisse, and W. Joosen, "A comparative study of workflow customization strategies: Quality implications for multi-tenant SaaS," J. Syst. Softw., vol. 144, pp. 423-438, 2018, doi: 10.1016/j.jss.2018.07.014.

[17]. H. Moens and F. De Turck, "Feature-Based Application Development and Management of Multi-Tenant Applications in Clouds," in Proc. 18th Int. Software Product Line Conf. (SPLC), vol. 1, pp. 72-81, 2014, doi: 10.1145/2648511.2648519.